The interaction of representation and reasoning

The Interaction of Representation and Reasoning *

Alan Bundy
University of Edinburgh
A.Bundy@ed.ac.uk

June 24, 2013

Abstract

Automated reasoning is an enabling technology for many applications of informatics. These applications include: verifying that a computer program meets its specification; enabling a robot to form a plan to achieve a task; and answering questions by combining information from diverse sources, e.g., on the internet.

How is automated reasoning possible? Firstly, knowledge of a domain must be stored in a computer, usually in the form of logical formulae. This knowledge might, for instance, have been entered manually, retrieved from the internet or perceived in the environment via sensors, such as cameras. Secondly, rules of inference are applied to old knowledge to derive new knowledge. Automated reasoning techniques have been adapted from logic, a branch of mathematics that was originally designed to formalise the reasoning of humans, especially mathematicians.

My especial interest is in the way that representation and reasoning interact. Successful reasoning is dependent on appropriate representation both of knowledge and of successful methods of reasoning. Failures of reasoning can suggest changes of representation. This process of representational change can also be automated. We will illustrate the automation of representational change by drawing on recent work in my research group.

1 The Automation of Reasoning

Reasoning can be automated by drawing on the formalisms provided by mathematical logic.

- *Knowledge is represented as a set of axioms in a logical theory.* For instance, the knowledge that $Elizabeth$ is female might be represented by the axiom: $Female(Elizabeth)$, where $Female$ is a *predicate* that given a person returns a truth value: true or false. The knowledge that Elizabeth is the mother of Charles might be represented by the axiom: $Mother(Charles) = Elizabeth$, where $Mother$ is a function that given a person returns their mother. Functions and predicates take a fixed number of *arguments* (aka inputs or parameters). In our examples $Female$ takes one argument, namely $Elizabeth$, and $Mother$ also takes one argument, namely $Charles$. They are, therefore, called *unary*. An example of a binary predicate, i.e., one taking two arguments, would be $Loves(Elizabeth, Charles)$.

  These examples are written in the language of first-order logic. A formal definition of this language is given in §1.2.

- *General purpose rules of inference are used to infer new theorems from axioms and previously proved theorems.* For instance, the *modus ponens* rule of inference is:

$$\frac{P, \quad P \implies Q}{Q}$$

*The research reported in this paper was supported by EPSRC Platform Grant EP/J001058/1, EPSRC project EP/J020524/1 and ONR Global project N62909-12-1-7012. I would like to thank my collaborators in the work surveyed here: Michael Chan, Jos Lehmann and Fiona McNeill. I would also like to thank Andriana Gkaniatsou, Fiona McNeill, Boris Mitrovic and two anonymous referees for feedback on an earlier version.

The two premises of this rule, which are written above the horizontal line, are that P is true and that if P is true then Q is true. The conclusion of the rule, which is written below the line, is that we can deduce that Q is true. The history of modus ponens goes back to antiquity. We will refer to the sets of axioms and rules as *logical theories*, but they are also sometimes called *ontologies* or *knowledge bases*. *Representation* is the process of formalising real world knowledge in a logical theory.

- *An automated theorem prover is used to apply rules of inference to axioms and old theorems to derive new theorems.* For instance, if our axioms include $Mother(Charles) = Elizabeth$ and $Mother(c) = m \implies Female(m)$ then a prover could use modus ponens to derive $Female(Elizabeth)$. Note that c and m are variables that range over people and that one of the operations involved in applying a rule of inference is to match variables, such as m, to constants, such as $Elizabeth$, and to instantiate such variables to the matching constant — or, more generally, to compound terms, such as $Mother(Charles)$. We will refer to the process of proving theorems as *reasoning*. Note that automated reasoning is often performed backwards: from a conjecture to be proved towards the axioms, i.e., the conclusion is matched to a conjecture and then an attempt is made to prove the premises as new conjectures or *sub-goals*. The process terminates with success when all the outstanding sub-goals can be matched to axioms.

1.1 Applications of Automated Reasoning

Automated reasoning is a key enabling technology in Informatics. It is applied, for instance, in the following areas:

Reasoning about Programs: To ensure that programs have been written correctly, it is possible to prove that they meet a specification of their behaviour. For instance, a program to sort a list of elements into alphabetic order can be formalised as a logical function and shown to output a list that is an ordered permutation of its input. This area is called *formal methods*. It can be used to show that a program is safe, secure and reliable for an infinite number of possible inputs, whereas program testing is inherently restricted to a finite set of inputs.

Forming Plans: Artificial agents, such as autonomous robots, must form plans to achieve the goals they are set. The initial state of the agent's environment and the actions it can perform can all be represented as axioms and the goal as a conjecture to be proved. A proof that the goal can be achieved by executing a sequence of actions can then be interpreted as the required plan.

Question Answering: Questions can be answered by combining information stored, for instance, on the Web or in databases. The known information can be represented as axioms and the question to be answered as a conjecture to be proved. The proof of this conjecture is a theorem that can be interpreted as showing how the answer to the question follows from already known information.

1.2 The Language of First-Order Logic

Axioms, theorems and other formulae of first-order logic are expressed in a formal language, which we describe in the following definitions.

Terms are used to describe objects of the domain. The domain is the collection of entities that a logical theory formalises.

Definition 1 (Terms) *A* term *is either constant, a variable or an n-ary*[1] *function applied to n terms. In our examples, Charles is a constant, c a variable, Mother a unary function and* $Mother(c)$ *is a compound term.*

[1] 'n-ary' is a generalisation of unary, binary, ternary, etc. to mean having n arguments. A function's *arity* is the number of its arguments.

We will adopt the notational convention that variable names start with a lower case letter, and constant, function and predicate names start with an upper case letter.

Propositions are used to describe individual facts and goals about these domain objects.

Definition 2 (Propositions) *A* proposition *is a truth value $\top$ (meaning true) or $\bot$ (meaning false) or an n-ary predicate applied to n terms. In our example Loves is a binary predicate and $Loves(Elizabeth, c)$ is a proposition.*

Formulae are constructed from propositions using logical connectives and quantifiers.

Definition 3 (Formulae) *A* formula *is a proposition, an n-ary connective applied to n formulae or a quantifier applied to a variable and a formula (usually containing that variable). We will use the following connectives:*

Negation: *$\neg P$ is true if and only if P is false.*

Conjunction: *$P \wedge Q$ is true if and only if both P and Q are true.*

Disjunction: *$P \vee Q$ is true if and only if either P or Q is true (or both).*

Implication: *$P \implies Q$ is true if and only if whenever P is true, then Q is true.*

and the following quantifiers:

Universal: *$\forall x.P(x)$ is true if and only if $P(c)$ is true for all c in the domain quantified over.*

Existential: *$\exists x.P(x)$ is true if and only if $P(c)$ is true for at least one c in the domain quantified over.*

We will use the term *expression* when we do not want to distinguish between terms, formulae, functions, etc.

When the domain is not uniform, but consists of different kinds of objects, it is often useful to distinguish the different kinds of objects to which expressions refer. Then quantification, for instance, can be restricted to range only over objects of one kind. This can be done by assigning sorts or types to expressions. We will encounter such a typed logic in §5. Example types are $\mathbb{R}$, the type of real numbers, $\mathbb{B}$ the type of truth values and $\mathbb{P}$, the type of people. We will use $e : \tau$ to assert that expression e has type τ, and we will use τ, possibly subscripted, to range over types. We can also assign types to functions, predicates, connectives, quantifiers, etc. For instance, $Mother : \mathbb{P} \mapsto \mathbb{P}$ means that $Mother$ is a function from people to people, where $\mapsto$ represents that one type of object is converted into another. Such type structures can get quite complicated. For instance, the quantifier $\forall$ can be considered to take a unary predicate and convert it to a truth value, which we can represent with the assignment $\forall : (\tau \mapsto \mathbb{B}) \mapsto \mathbb{B}$. Note the use of the type variable τ, which indicates that the argument of the unary predicate can be of any type. When something has type variables in its type, it is called *polymorphic*.

A logical theory $\mathcal{T}$ can be represented as a pair $\langle Sig(\mathcal{T}), Ax(\mathcal{T})\rangle$, where $Sig(\mathcal{T})$ is the *signature* of the ontology and $Ax(\mathcal{T})$ is a set of axioms. $Sig(\mathcal{T})$ is a set of declarations of the types of the various constants, functions and predicates which are used in formulae of $\mathcal{T}$. It can be used as the basis of a recursive definition of the language of the theory, i.e., we can define the terms, formulae, etc. in terms of the constants, functions and predicates declared in the signature. For instance, we might define a theory $Family$ with:

$$\begin{aligned} Sig(Family) &= \{Charles : \mathbb{P}, Elizabeth : \mathbb{P}, Mother : \mathbb{P} \mapsto \mathbb{P}, Female : \mathbb{P} \mapsto \mathbb{B}\} \\ Ax(Family) &= \{Mother(Charles) = Elizabeth, Mother(c) = m \implies Female(m)\} \end{aligned}$$

We will write $\mathcal{T} \vdash \mathcal{F}$ to mean that $\mathcal{F}$ is a theorem of $\mathcal{T}$. For instance, $Family \vdash Female(Elizabeth)$.

1.3 The Resolution Rule of Inference

A common rule of inference used in the automation of inference in first-order logic is *resolution* [Bachmair & Ganzinger, 2001]. This rule is described in Definition 6. Resolution uses *clausal form*, *substitution* and *unification*, which are described in §1.3.1 and §1.3.2.

1.3.1 Clausal Form

Binary resolution applies to two clauses and derives a third clause. The axioms of a first-order theory are transformed into an equivalent conjunction of clauses. The conjecture to be proved is negated and also transformed into a conjunction of clauses. These two conjunctions are themselves conjoined and resolution is repeatedly applied to them.

Definition 4 (Literals and Clauses) *A* literal *is either a proposition or a negated proposition. A* clause *is a disjunction of literals. The empty clause is represented by* $\square$*, which is logically equivalent to* $\bot$ *(false).*

Resolution proofs work by *reductio ad absurdum*[2], i.e., the conjecture is negated and the aim is to infer the empty clause $\square$ (false). The original conjecture is then proved.

To transform a formula into *clausal form*, it must be reorganised so that:

- $\neg$ only appears in front of propositions, not compound formulae.
- $\vee$ only disjoins literals.
- $\wedge$ only conjoins clauses.
- $\forall$ and $\exists$ are eliminated in a process called *skolemisation*. Universally quantified variables are regarded as implicitly universal and the quantifier is removed. The previous universally quantified variable is now called a *free variable*. Existential quantified variables are transformed into new *skolem functions*, whose arguments are just those universal variables that are quantified before the existential one.

For instance, the definition of a continuous function is:

$$\forall x.\forall \epsilon > 0.\exists \delta > 0.\forall b.\ (|b| < \delta \implies |f(x+b) - f(x)| < \epsilon) \tag{1}$$

Skolemising this formula gives:

$$\epsilon > 0 \implies (\Delta(x,\epsilon) > 0 \wedge (|b| < \Delta(x,\epsilon) \implies |f(x+b) - f(x)| < \epsilon)) \tag{2}$$

where Δ is a new, binary skolem function. Notice that Δ has x and ϵ as arguments, but not b, because, reading left to right, x and ϵ are quantified before δ, but b is quantified afterwards.

This skolemised formula can now be transformed into two clauses:

$$\begin{aligned} &\neg(\epsilon > 0) \vee \neg(\Delta(x,\epsilon) > 0) \wedge \\ &\neg(\epsilon > 0) \vee \neg(|b| < \Delta(x,\epsilon)) \vee (|f(x+b) - f(x)| < \epsilon) \end{aligned} \tag{3}$$

Suppose that instead of using formula (1) as an *axiom* to assert the continuity of a function, we wanted to prove the continuity of a function as a *theorem*. Formula (1) would now have to be negated before putting it into clausal form.

$$\neg\forall x.\forall \epsilon > 0.\exists \delta > 0.\forall b.\ (|b| < \delta \implies |f(x+b) - f(x)| < \epsilon)$$

We use the equivalences:

$$\neg\forall x.P \iff \exists x.\neg P \quad \text{and} \quad \neg\exists x.P \iff \forall x.\neg P$$

[2] `http://en.wikipedia.org/wiki/Reductio_ad_absurdum`

to move the $\neg$ inside the quantifiers to give:

$$\exists x.\exists \epsilon > 0.\forall \delta.\exists b.\ \neg(|b| < \delta \implies |f(x+b) - f(x)| < \epsilon)$$

Skolemising this formula then gives:

$$E > 0 \wedge (\delta > 0 \implies (|B(\delta)| < \delta \implies |f(X + B(\delta)) - f(X)| < E)) \tag{4}$$

Compare formula (4) with (2). Note how the free variables and skolem functions swap roles as a result of negation.

This skolemised formula can now be transformed into three clauses:

$$\begin{aligned} &E > 0 \wedge \\ &\neg(\delta > 0) \vee (|B(\delta)| < \delta) \wedge \\ &\neg(\delta > 0) \vee \neg(|f(X + B(\delta)) - f(X)| < E) \end{aligned} \tag{5}$$

Compare the clauses (3) with (5). Note how the roles of $\vee$ and $\wedge$ are swapped and how negated literals become unnegated and vice versa.

1.3.2 Substitutions and Unification

A substitution is used to create an instance of a term or formula. It consists of a set of instructions to replace variables with terms.

Definition 5 (Substitution) *A* substitution *is a set of pairs x_i/s_i, for $1 \leq i \leq n$, where the x_i are distinct variables and the s_i are terms that contain none of the x_j for $1 \leq j \leq n$.*

We will use σ, possibly subscripted, to represent substitutions.

We write $t\sigma$ to mean the term t with the substitution σ applied to it. For instance, $Loves(x, y)\{x/Elizabeth, y/Charles\}$ evaluates to $Loves(Elizabeth, Charles)$.

Unification is an algorithm for creating substitutions. Given two terms or propositions, say s and t, it will, if it is possible, create a substitution σ, called a *unifier*, such that $s\sigma$ and $t\sigma$ are identical. It also creates the most general such unifier, i.e., it does not instantiate any variable strictly more than necessary to make the two terms/propositions identical. For instance, the most general unifier of $Loves(Elizabeth, y)$ and $Loves(u, v)$ is $\{u/Elizabeth, v/y\}$. The substitution $\{u/Elizabeth, v/Charles, y/Charles\}$ would also be a unifier, but not the most general one in this case. If variations in the names of variables are ignored, most general unifiers are unique. A version of the unification algorithm adapted from [Baader & Snyder, 2001][p455] is described in Table 1.

We are now in a position to define the *binary resolution rule*.

Definition 6 (Binary Resolution)

$$\frac{P_1 \vee C_1, \quad \neg P_2 \vee C_2}{(C_1 \vee C_2)\sigma}$$

where the P_i are propositions, the C_i are clauses and σ is the most general unifier of P_1 and P_2.

The unification algorithm finds a substitution, σ, that makes $P_1\sigma$ identical to $P_2\sigma$. A new clause is then formed by disjoining the remaining literals C_1 and C_2, then applying σ to all the literals in the new clause. Modus ponens is a special case of resolution. An example of binary resolution is:

$$\frac{Mother(Charles) = Elizabeth, \quad \neg Mother(c) = m \vee Female(m)}{Female(Elizabeth)}$$

where P_1 is $Mother(Charles) = Elizabeth$, P_2 is $Mother(c) = m$ and σ is $\{c/Charles, m/Elizabeth\}$.

Case	**Before**	**Condition**	**After**
Trivial	$s \equiv s \wedge E; \sigma$		$E; \sigma$
Decomp	$f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n) \wedge E; \sigma$		$s_1 \equiv t_1 \wedge \ldots \wedge s_n \equiv t_n \wedge E; \sigma$
Clash	$f(s_1, \ldots, s_m) \equiv g(t_1, \ldots, t_n) \wedge E; \sigma$	$f \neq g \vee m \neq n$	fail
Orient	$t \equiv x \wedge E; \sigma$		$x \equiv t \wedge E; \sigma$
Occurs	$x \equiv s \wedge E; \sigma$	$x \in \mathcal{V}(s) \wedge x \neq s$	fail
Var Elim	$x \equiv s \wedge E; \sigma$	$x \notin \mathcal{V}(s)$	$E\{x/s\}; \sigma \oplus \{x/s\}$

Table 1: **Standard Unification Algorithm:** Each row of the table represents a transformation rule: if the current unification problem matches the Before column and any Condition is satisfied, then it is transformed into the After column. A unification problem is expressed as $e_1 \equiv e_2; \sigma$ where the e_i are the two expressions to be unified and σ is the unifier formed so far, initially the empty substitution, which is the empty set. $\oplus$ is the composition operation for substitutions. f and g are functions or predicates. t is a non-variable term and s is any term. $\mathcal{V}(s)$ is the set of free variables in s. s_i and t_j are terms. $m, n \geq 0$, x and y are distinct variables. E is a, possibly empty, conjunction of unification problems. The algorithm terminates with success and returns σ when the Before state is $\top; \sigma$, where $\top$ is the empty conjunction.

Binary resolution needs to be supplemented with the *factorisation rule*[3]. Factorisation merges several literals in a clause into one by applying their most general unifier to the clause, so that these literals become identical. The combination of binary resolution and factorisation is *complete*, meaning that if a conjecture is provable from some axioms, then these two rules will find a refutation, i.e., will prove $\square$. If, however, the conjecture is not provable, then:

- either a stage will be reached at which no further rules are applicable, but $\square$ has not been proved,
- or the rules may keep on finding new clauses without terminating.

Because of this potential for non-termination, resolution is *semi-decidable*, i.e., it may never decide the question of whether a conjecture is a theorem, since the fact that it has not terminated yet does not mean that it won't eventually.

2 Reasoning Failures and Possible Repairs

We are interested in how reasoning failures can trigger representational change. Moreover, we focus on *conceptual* change, i.e., not just the deletion or addition of axioms in a representation, but a change in the *language* in which the axioms are expressed. Typical language changes include: the splitting of a function (or predicate) into two or more functions (or predicates); the merging of two distinct functions (or predicates) into one; and a change in the number of arguments that a function (or predicate) takes, e.g., by removing or adding one or more of them.

We will consider the following kinds of reasoning failure:

Proof of a false conjecture: If a false conjecture can be proved in a logical theory, then it can be repaired either by removing one or more axioms or rules, or by changing the language in which one or more of them are expressed. For instance, suppose it is possible to prove both $Capital(Japan) = Tokyo$ and $Capital(Japan) = Kyoto$[4], where $Capital$ is a function that takes a country and returns its capital city. If either of these theorems is an axiom, then the

[3] Alternatively, factorisation can be merged with binary resolution to give *full resolution*, where unification is simultaneously used to unify one or more literals in each parent clause and then they are resolved together.

[4] See [Gkaniatsou *et al*, 2012] for a discussion of how this error has arisen in practice.

representation can be repaired by removing one of them. Alternatively, since Kyoto used to be the capital of Japan, we could retain this information by introducing a new function $WasCapital$ and a replacement axiom $WasCapital(Japan) = Kyoto$. Or we could add an additional time argument to $Capital$, e.g., with replacement axioms $Capital(Japan, Now) = Tokyo$ and $Capital(Japan, Past) = Kyoto$[5].

Failure to prove a true conjecture: If we can't prove something that is true then we can repair the representation by adding a new axiom or rule, or by changing the language. For instance, suppose that $Female(Elizabeth)$ is not a theorem. Perhaps the axiom $Mother(Charles) = Elizabeth$ is missing, so we could add it. Alternatively, perhaps the axioms we were hoping to derive $Female(Elizabeth)$ from are $Mother(Charles) = Elizabeth$ and $MumOf(c) = m \implies Female(m)$, where $Mother$ and $MumOf$ are two different names for what was intended to be the same function. This might happen, for instance, if the representation has been formed by merging two different ones, or if multiple authors of a representation have been working at cross purposes as to the correct name for this function. The representation could then be repaired by renaming, say, $MumOf$ to $Mother$, or vice versa. Figure 1[6] describes how an everyday object can be formalised in two different ways, and making the correct choice between them is crucial for success.

Reasoning runs out of resources: Reasoning is computationally expensive. Multiple rules are often applicable to each (sub-)goal, perhaps in multiple ways, and each may need to be tried during a proof attempt. The number of alternative proof attempts can grow exponentially (or worse) in the length of the proof to be discovered. So reasoning can fail due to exhaustion of resources, e.g., the time or the storage available. As Pólya has emphasised, however, successful problem solving is very dependent on finding an appropriate representation [Pólya, 1945]. In a better representation a more succinct proof may be possible or one requiring less search. Figure 2 describes the Mutilated Checker Board[7] Problem, in which a change of representation makes a dramatic difference to the efficiency of problem solving and the understandability of the solution.

Representational changes of the kind illustrated above are frequently manually applied to computer representations as part of development and maintenance. Automatic addition and removal of axioms has also been implemented in the fields of abductive reasoning and belief revision, respectively. These two fields are briefly discussed in §8. Automated conceptual/language change is less common, but has been pioneered in my research group, examples of which will be given below.

3 Why is Automated Representational Change Important?

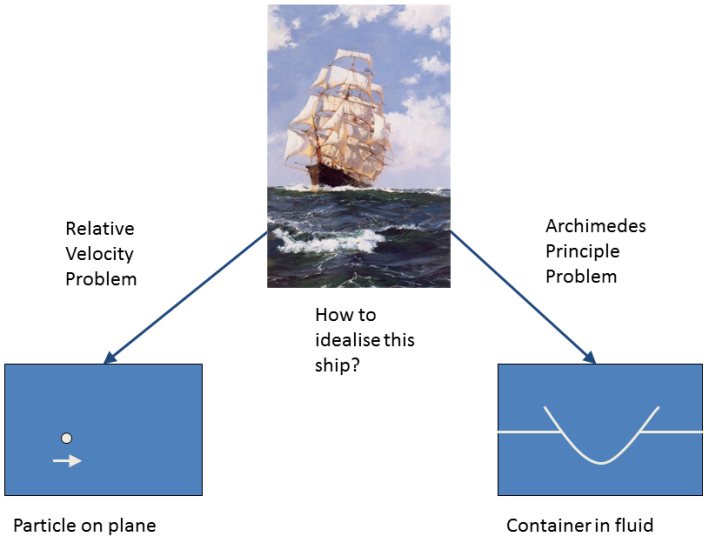
Automated agents are becoming more of a feature of high-tech applications. They range from industrial, domestic and military robots, to purely software agents in personal mobile devices, to programs used in finance and commerce, e.g., to buy and sell financial instruments. Such agents require representations of their environment, including representations of the representations of other agents, to form plans, negotiate transactions, answer questions, etc. Until recently, such agents worked in a relatively stable environment and on a relatively stable task, which meant that their representations could also be stable. Typically, these representations were built manually by the designer of the agent and any changes required would also be handled manually during upgrades.

No universal representation is possible. The world is infinitely rich and any representation is an approximation of that richness. The designer of a logical theory must find a sweet spot between

[5] Or we could use a more sophisticated representation of time.

[6] "Racing Home - The Cutty Sark" by Montague Dawson: reproduced with permission from `http://one1more2time3.wordpress.com/tag/maritime-art/` ©montague dawson.

[7] This problem was originally posed by an American. In Britain we would call it a chess board or draughts board.



In Physics problem solving, real world objects, such as ships or people, must be idealised as standard physical concepts. Students of Physics and Applied Mathematics are taught to do this by first recognising the problem type. For instance, in a relative velocity problem, a ship might be idealised as a particle on a horizontal plane, representing the sea. In an Archimedes Principle problem, on the other hand, the ship might be idealised as a container floating in a fluid. Appropriate idealisation is the key to successful problem solving. In real life problem solving, of course, problems do not fall so neatly into types and idealisation can be a challenging problem.

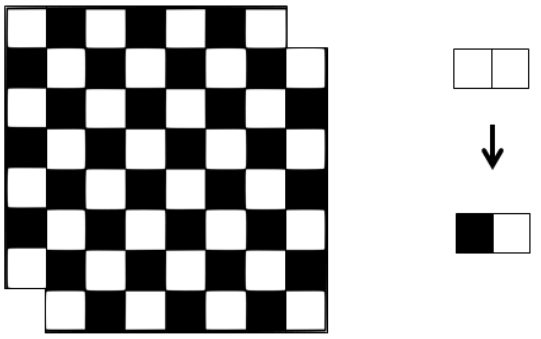
Figure 1: Representation is Key to Problem Solving

expressivity and efficiency; that is, the theory must be rich enough for the agent to perform its task, but simple enough for automated reasoning to be tractable. The agent's representation will be tuned to its role.

Many future applications of automated agents will, however, require them to adapt to a changing environment and a changing role. Representational change will be required:

- to accurately model changes in the environment;
- to ensure that reasoning is appropriate for any new reasoning task the agent is set; and
- to enable communication with other agents which are using different representations of the environment.

Sometimes the need for representational change will be sufficiently urgent that it cannot wait on the designer to make manual changes, but must be automatic and dynamic, i.e., implemented while the agent is still reasoning [McNeill & Bundy, 2007]. Consider, for instance, emergency response. In an emergency, multiple agencies must collaborate to solve problems, some of which are unforeseen. In a flood, police, health services, fire brigade, the military, local government, the meteorological office, various NGOs, etc., must collaborate in a rapidly changing situation. This collaboration will include the sharing of information stored digitally in different formats. Automated and dynamic alignment and updating of these different information sources is essential. Failures due to lack of effective communication are frequently cited in post-emergency reports, so this is a real problem, which is likely to increase in importance as the rate of emergencies increases as a side-effect of climate change.



A checker board is mutilated by removing one square from each end of a diagonal. Is it possible to cover the above mutilated checker board completely with non-overlapping dominoes, each of which covers exactly two adjacent squares? One could exhaustively explore all possible coverings. A computer could be programmed to do this and with modern computers this could be done without exhausting its resources, but the answer, which is no, is unsatisfyingly brute force and uninformative. A much more satisfying and efficient solution is to colour the dominoes half black and half white then to insist, without loss of generality, that the colours of the dominoes match those of the squares they cover. Before mutilation, the checker board had exactly the same number of black squares as white ones, now it has two fewer black ones. Since each domino covers exactly one white and one black square, then even the best covering must leave two uncovered white squares. This representation provides a much clearer explanation of why a complete covering is impossible. This problem was popularised within automated reasoning in [McCarthy, 1964].

Figure 2: The Mutilated Checker Board

4 Repairing an Agent's Representation of Other Agents

The ORS System (Ontology[8] Repair System) [McNeill & Bundy, 2007] uses a planning agent's representations of the services provided by service-providing agents to construct a plan composed of those services. For instance, the planning agent might plan a holiday by composing the services provided by airlines, hotels, tour providers, etc. Its representations of these other agents may, however, be inaccurate. If so, then its plan may fail to execute, for instance, a service-providing agent may refuse to provide its expected service. ORS then tries to diagnose the cause of the failure, repair the planning agent's representations to be a better match to those of the service-providing agents, then re-plan: repeating this process until either a plan succeeds or it is unable to diagnose the cause of failure and/or repair the faulty ontology.

The services provided by the service-providing agents are represented by STRIPS-like planning operators [Fikes & Nilsson, 1971]. These describe the preconditions under which an agent will provide a service and the effects of doing so. ORS's diagnosis process works by considering the *execution* of the plan as a *failed* proof of a true conjecture: that is, it assumes that there is a plan that will achieve its goals, but that it has failed to find it. It uses clues from the failed execution of the unsuccessful plan to diagnose the errors in its representation.

One such clue is a *surprising question*. The planning agent expects to be asked certain questions during the course of the execution of its plan. These questions might arise, for instance, when a service-providing agent is checking the preconditions of its service. Some of these preconditions

[8]The term 'ontology' is usually preferred over 'logical theory' in the field of software agents on the internet.

it can check itself, but some have to be directed to the planning agent, which expects these questions. Communication between the agents takes the form of logical formulae, sometimes containing variables that the recipient is asked to instantiate and return the values of. For instance, a hotel agent might ask if the planning agent has the deposit for a room. The planning agent might expect this question to take the form $Money(£200)$. Suppose that this question is not asked, but that the surprising question $Money(£200, Credit\_Card)$ is asked instead. ORS can make a fuzzy match between the expected and the surprising question and infer that the planning agent has a unary version of $Money$ but that the hotel agent has a binary one. Moreover, the second argument is $Credit\_Card$, i.e., the hotel requires to be paid the deposit by credit card, whereas the planning agent thought any form of payment was acceptable.

ORS must now update the planning agent's ontology by changing its unary predicate $Money$ to a binary one. In the precondition of the hotel's service the second argument of this predicate must be $Credit\_Card$. The planning agent will now replan with this new representation and try to execute the new plan. Sometimes multiple cycles of this plan, diagnose and repair process are required before a successful plan is formed.

5 Repairing Physics Theories

The GALILEO System (Guided Analysis of Logical Inconsistencies Leads to Evolved Ontologies) suggests repairs to faulty theories of Physics when they are contradicted by reliable experimental evidence [Lehmann *et al*, 2012]. GALILEO has one ontology to represent the theory and another to represent the conflicting experimental evidence.

GALILEO uses a number of what we call *ontology*[9] *repair plans* (ORPs). Each ORP consists of a trigger formula and some representation repair instructions: when the trigger formula can be proved, the repair is executed. The trigger formula describes theorems of two or more ontologies, which taken together are in conflict, e.g., are inconsistent. The most commonly applicable ORP is Where's My Stuff (WMS). Its trigger formula is:

$$\begin{aligned} \mathcal{O}_x &\vdash f(\mathit{stuff}) = v_x \\ \mathcal{O}_y &\vdash f(\mathit{stuff}) = v_y \\ \mathcal{O}_{arith} &\vdash v_x \neq v_y \end{aligned}$$

where the $\mathcal{O}_i$ are distinct ontologies: $\mathcal{O}_x$ might be the representation of a Physics theory; $\mathcal{O}_y$ that of some conflicting experimental evidence; and $\mathcal{O}_{arith}$ the representation of a theory of arithmetic. Some attribute f of some *stuff* is predicted to take the value v_x, but experiment shows it to have a different value v_y. Figure 3[10] describes an application of WMS.

First-order logic is not ideal for representing many aspects of Physics as it is often desirable to have variables ranging over functions and to have functions that take other functions as arguments. Consider, for instance, differential calculus, where the differentiation operation is naturally represented as a function that converts one unary function into another. For this reason, logical theories in GALILEO are represented in higher-order logic, which permits function variables and functions of functions.

Associated with the WMS trigger is a particular form of repair. New versions of $\mathcal{O}_x$ and $\mathcal{O}_y$ are constructed, which we will call $\nu(\mathcal{O}_x)$ and $\nu(\mathcal{O}_y)$, respectively. Recall from §11.1 that an ontology $\mathcal{O}$ can be represented as a pair of sets of the type declarations (the signature) and the axioms: $\langle Sig(\mathcal{O}), Ax(\mathcal{O})\rangle$. For instance, the set of type declarations $\{\mathit{stuff}{:}\tau,\ v_x{:}\tau', f : \tau \mapsto \tau'\}$ means that *stuff* is of type τ, v_1 is of type τ' and f is a function from things of type τ to things of type τ'. Typical types in GALILEO are be real numbers, graphs, galaxies of stars, etc.

[9] The term 'ontology' is used to avoid confusion between logical and physical theories.

[10] The Andromeda Galaxy photo: Robert Gendler - reproduced with permission from `www.robgendlerastropics.com`. The graph: adapted from `http://en.wikipedia.org/wiki/File:GalacticRotation2.svg` and published by the Royal Society under the terms of the Creative Commons Attribution-Share Alike 2.0 license `http://creativecommons.org/licenses/by-sa/2.0/uk/deed.en`.

The signatures for the new ontologies are updated in terms of those of the old as follows:

$$\begin{array}{lcl} Sig(\nu(\mathcal{O}_x)) & ::= & \{\mathit{stuff}_{vis}{:}\tau, \mathit{stuff}_{invis}{:}\tau\} \cup Sig(\mathcal{O}_x) \\ Sig(\nu(\mathcal{O}_y)) & ::= & \{\mathit{stuff}_{vis}{:}\tau\} \cup Sig(\mathcal{O}_y) \setminus \{\mathit{stuff}{:}\tau\} \end{array}$$

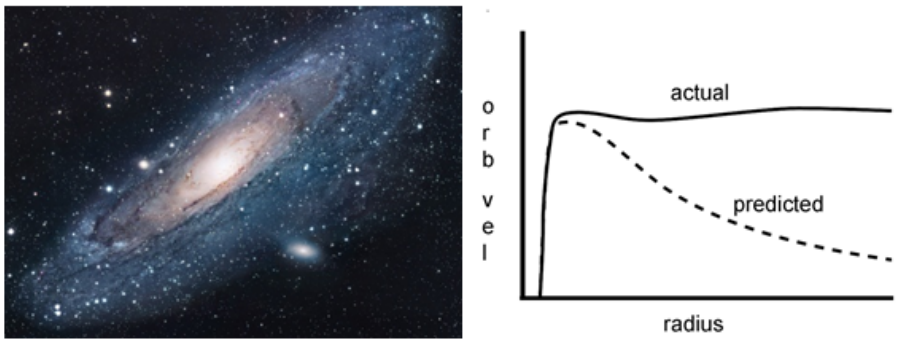
That is, two new objects, stuff_{vis} and stuff_{invis} are created with the same type as *stuff*. The idea is that *stuff* will now be divided into visible and invisible stuff. They are both added to the signature of $\mathcal{O}_x$, and stuff_{vis} is used to replace *stuff* in $\mathcal{O}_y$. So in $\mathcal{O}_x$ we can refer to visible, invisible and total stuff, but in $\mathcal{O}_y$ we only refer to visible stuff.

The axioms for the new ontologies are updated in terms of those of the old as follows:

$$\begin{array}{lcl} Ax(\nu(\mathcal{O}_x)) & ::= & \{\ \mathit{stuff}_{invis} ::= \mathit{stuff} -_\tau \mathit{stuff}_{vis}\ \} \ \cup\ Ax(\mathcal{O}_x) \\ Ax(\nu(\mathcal{O}_y)) & ::= & \{\phi\{\mathit{stuff}/\mathit{stuff}_{vis}\} \mid \phi \in Ax(B)\ \} \end{array}$$

$\nu(\mathcal{O}_x)$ contains all the axioms of $\mathcal{O}_x$ but also has a new axiom that connects the three kinds of stuff: invisible stuff is defined as the difference between total stuff and visible stuff. $-_\tau$ is a form of subtraction that is appropriate for objects of type τ. For instance, if τ is the real numbers then $-_\tau$ will be ordinary subtraction, but if τ is a set then $-_\tau$ will be set minus. $\nu(\mathcal{O}_y)$ contains all the axioms of $\mathcal{O}_y$, but with every occurrence of *stuff* replaced with stuff_{vis}.

The new version of the trigger formula no longer suggests a conflict. $\nu(\mathcal{O}_y) \vdash f(\mathit{stuff}_{vis}) = v_y$ is now an observation only about visible stuff, so does not contradict $\nu(\mathcal{O}_x) \vdash f(\mathit{stuff}) = v_x$, which is about total stuff.



An anomaly has been detected in spiral galaxies in the relationship between the orbital velocities of the stars and their distance to the centre of the galaxy. The graph on the right plots orbital velocity against distance to the centre. The dotted line marked A shows the theoretical prediction of this relationship using Newtonian mechanics. The continuous line marked B shows the experimental observations. This situation matches the WMS trigger, with the galaxy matching stuff, the function that takes a galaxy and generates the graph matching f, and the graphs A and B matching v_y and v_x, respectively. The repair is to split the stuff concept: stuff_{vis} represents the visible galaxy; stuff_{invis} represents a halo of dark matter surrounding the galaxy; and stuff is recycled to represent the sum of these two. When the dark matter halo is taken into account, the revised predictive A graph matches the observed B graph, thus resolving the anomaly.

Figure 3: WMS Example: Dark Matter

6 Repairing Faulty Mathematics Proofs

In [Lakatos, 1976][Appendix I], Lakatos describes Cauchy's faulty proof that the limit of a convergent series of continuous functions is itself continuous. Fourier analysis had provided a counter-example to this faulty theorem. One can give a convergent series of (continuous) sine functions whose limit is the (discontinuous) square wave. For a long while, even though this counter-example was known, the error in the proof lay undiscovered. Its eventual repair was to replace 'convergence' with '*uniform* convergence' in the theorem's premise. The only difference between these two concepts is the quantifier order.

Convergence: $\forall x.\forall \epsilon > 0.\exists m.\forall n \geq m.\ |\sum_i f_i(x)| < \epsilon$

Uniform Convergence: $\forall \epsilon > 0.\exists m.\forall x.\forall n \geq m.\ |\sum_i f_i(x)| < \epsilon$

Note that $\forall x$ is moved from being the first quantifier in convergence to being the third in uniform convergence.

In [Bundy, 1985] we noted that the error in Cauchy's proof can, when represented as a resolution proof attempt[11], be described as an *occurs check* error. That is, there is an unsuccessful attempt to resolve:

$$y \geq y \quad \text{with} \quad n \geq M(x + B(\Delta(E/3, x, n)), E/3)$$

where M, E, B and Δ are skolem functions arising from the existentially quantified variables m, ϵ, b and δ in the definitions of convergence and continuity. This resolution step will fail due to an application of step *Occurs* in the unification algorithm described in Table 1, since the *Occurs* condition $n \in \mathcal{V}(M(x+B(\Delta(E/3, x, n)), E/3))$ is true. The context in which this failed resolution step arises is described in Figure 4.

Let $f(x) = \sum_i f_i(x)$ be the function that is defined by a convergent series of the continuous functions f_i. The goal is to prove that f is continuous, i.e., that:

$$\forall x.\forall \epsilon > 0.\exists \delta > 0.\forall b.\ |b| < \delta \implies |f(x+b) - f(x)| < \epsilon$$

When this goal is negated and converted into clausal form, we saw in §1.3.1 clause set (5) that one of the clauses produced is:

$$\neg(\delta > 0)) \vee \neg(|f(X + B(\delta)) - f(X)| < E) \tag{6}$$

From the definition of convergence of the series we can deduce the clause:

$$\begin{aligned} &\neg(\epsilon > 0) \vee \neg(|b| < \Delta(\epsilon, x, n)) \vee \neg(n \geq M(\epsilon, x)) \\ &\quad \vee \neg(n \geq M(\epsilon, x + b)) \vee |f(x + b) = f(x)| < 3.\epsilon \end{aligned} \tag{7}$$

After rewriting E to $3.E/3$, clauses (6) and (7) can be resolved on their last literals. The literals in the resulting clause can all be resolved away, except for:

$$\neg(n \geq M(x + B(\Delta(E/3, x, n)), E/3))$$

As discussed in the main text, the attempt to resolve this literal with the axiom $y \geq y$ fails due to an occurs check failure.

Figure 4: A Fragment of Cauchy's Faulty Proof

[11] Of course, resolution, unification and skolem functions had not been invented when Cauchy produced this faulty proof. Our version of the faulty proof is a modern reconstruction. In fact, the diagnosis of such faulty proofs led to Frege's invention of first-order logic and thus, indirectly, to resolution, etc.

One way to repair this faulty proof is to enable the blocked resolution step to succeed. For this it is necessary to remove the embedded n from $M(x + B(\Delta(E/3, x, n)), E/3)$. Since this unwanted n is embedded in three skolem functions: M, B and Δ this can be done by reordering the quantifiers in the definitions of convergence or continuity. For instance, m has two arguments because it is preceded by two universally quantified variables: x and ϵ. We want to remove its first argument, which is supplied by the $\forall x$. This can be done by moving $\forall x$ after $\exists m$ in the definition of convergence. This gives the desired definition of uniform convergence. Now the previously faulty resolution step is replaced by:

$$\text{resolve } y \geq y \quad \text{with} \quad n \geq M(E/3)$$

which succeeds since $n \notin \mathcal{V}(M(E/3))$.

The variable n was, though, originally nested within two other skolem functions: B and Δ, so two other repairs are also suggested.

- B is a skolem function that arises from the definition of continuity:

  $$\forall x.\forall \epsilon > 0.\exists \delta > 0.\forall b.\ |b| < \delta \implies |f(x + b) - f(x)| < \epsilon$$

  when it is negated, since the originally universally quantified b becomes existentially quantified when the $\neg$ is moved inwards (see formula (4)). To effect this repair we need to move $\exists \delta > 0$ after $\forall b$, i.e.,

  $$\forall x.\forall \epsilon > 0.\forall b.\exists \delta > 0.\ |b| < \delta \implies |f(x + b) - f(x)| < \epsilon$$

  This, however, is a trivial property that holds for all functions, so does not suggest a sensible corrected theorem.

- Δ is a skolem function that arises from the definition of continuity in the following form:

  $$\forall n \in \mathbb{N}.\forall x.\forall \epsilon > 0.\exists \delta > 0.\forall b.\ |b| < \delta \implies |S(n, x + b) - S(n, x)| < \epsilon$$

  where $S(n, x)$ is the finite sum of the f_i up to n, i.e., $\sum_i^n f_i(x)$. To effect this repair we need to move the $\forall n \in \mathbb{N}$ after the $\exists \delta > 0$ to get:

  $$\forall x.\forall \epsilon > 0.\exists \delta > 0.\forall n \in \mathbb{N}.\forall b.\ |b| < \delta \implies |S(n, x + b) - S(n, x)| < \epsilon$$

  This concept was independently discovered by Ascoli, who called it *equi-continuity* [Ascoli, 1883]. It leads to a different and interesting repair of Cauchy's theorem in which the f_i are equi-continuous. Ascoli also proved a theorem about equi-continuity, but not this one.

7 Reformation: A Domain-Independent Algorithm for Representation Language Repair

In §4, §5 and §6 we have described domain-specific algorithms for repairing the languages in which representations are formalised. Is there a general-purpose algorithm that subsumes them all? We are developing such an algorithm, which we call *reformation* [Bundy & Mitrovic,].

Reformation is designed to deal with two of the three types of failure described in §2: the proof of a false conjecture and the failure to prove a true conjecture. It is adapted from the unification algorithm, described in Table 1. The unification rules have been re-ordered into complementary pairs. The members of each pair share the same Before pattern, but have opposite Conditions: with one member a success case and the other a failure case. The algorithm deals both with situations in which a successful unification must be blocked and those in which an unsuccessful one must be unblocked. Blocking works by switching a success case to a failure one; unblocking works by switching a failure case to a success one. The algorithm is described as a set of transformation rules in Table 2.

Case	**Input Problem**	**Condition**	**Block**	**Unblock**
CC_s	$f(s_1,\ldots,s_m) \equiv g(t_1,\ldots,t_n) \wedge E$	$f = g \wedge m = n$	Make $f \neq g \vee m \neq n$ $\bigvee_{i=1}^{n}$ Block $s_i \equiv t_i$ $\vee$ Block E	$\bigwedge_{i=1}^{n}$ Unblock $s_i \equiv t_i$ $\wedge$ Unblock E
CC_f		$f \neq g \vee m \neq n$	No action	Make $f = g \wedge m = n$ $\bigwedge_{i=1}^{n}$ Unblock $s_i \equiv t_i$ $\wedge$ Unblock E
VC_f	$x \equiv t \wedge E$ or $t \equiv x \wedge E$	$x \in \mathcal{V}(t)$	No action	Make $x \notin \mathcal{V}(t)$ $\wedge$ Unblock $E\{x/t\}$
VC_s		$x \notin \mathcal{V}(t)$	Make $x \in \mathcal{V}(t)$ $\vee$ Block $E\{x/t\}$	Unblock $E\{x/t\}$

Table 2: **The Reformation Algorithm**: The notational conventions are as for Table 1. We have reorganised the unification algorithm rows so that the two compound/compound cases and the two variable/compound cases are adjacent, which suggests repairs that switch one member of each pair for the other. No repair is possible in the variable/variable case, so it is omitted. The *Trivial* step is deliberately omitted, as it skips over CC_s steps that reformation might want to block. The *Orient* step is subsumed into VC_f and VC_s. Reformation is given a parameter to tell it whether to attempt to block or unblock the unification. The *Block* column gives a disjunction of repair suggestions, any of which will block the unification. The *Unblock* column gives a conjunction of repair suggestions, all of which are required to unblock the unification. The *Block* and *Unblock* commands are recursive calls to reformation with the appropriate parameter setting.

Note that reformation is completely automatic, i.e., no human intervention is required for it to suggest repairs. Both blocking and unblocking are guaranteed to succeed, i.e., they always suggest one or more repairs that will either block unification, so that it fails, or unblock unification, so that it succeeds. Its efficiency is essentially the same as for unification, i.e., it is exponential in the size of the input problem. It is well known that the occurs check is the source of the exponential runtime, but linear versions of unification have been devised, so there is potential to adapt these more efficient versions of unification to construct a more efficient version of reformation. Unification is used in resolution provers that generate millions of intermediate clauses in tractable runtimes, so its efficiency does not seem to be a practical problem.

What is a practical problem, however, is the huge number of repairs that reformation suggests. For instance, a large resolution proof can be blocked in several ways at every step of unification in every application of resolution. Heuristics are needed to decide which blocking repair suggestion to adopt. For unblocking, the situation can be much better. If we have a candidate faulty proof, then this will typically only be blocked at a few resolution steps, so only these steps need to be repaired, leading to only a handful of unblocking repair suggestions. This is the case, for instance, for the example described in §6.

Reformation can be successfully applied to the examples in §4, §5 and §6.

ORS: In §4 we described a failed unification between $Money(£200)$ and $Money(£200, Credit\_Card)$. When reformation is asked to unblock this unification, transformation rule CC_f is fired. It suggests making the arities of the two occurrences of $Money$ be the same. One way to realise this repair is to add an additional argument to the first occurrence to make it identical to the second one.

Galileo: In §5 the inconsistency can be formalised as arising from a successful but unwanted unification between two occurrences of $f(stuff)$. When reformation is asked to block this unification, transformation rule CC_s is fired both on $f(stuff) \equiv f(stuff)$ and then, recursively, on $stuff \equiv stuff$. It makes various suggestions, one of which is to distinguish the two occurrences of *stuff* by renaming one of them.

Faulty proofs: In §6 we described a failed unification between $y \geq y$ and $n \geq M(x+B(\Delta(E/3, x, n)), E/3)$. When reformation is asked to unblock this unification, transformation rule VC_f is fired. It suggests removing n from $M(x + B(\Delta(E/3, x, n)), E/3)$. As we have seen, this can be done in three[12] different ways, two of which lead to significant and interesting theorems.

More generally, our experience with reformation so far is that it has always found the theory repair that we had hoped to find, e.g., the repair proposed as desirable by some third party expert. It also, however, finds a huge number of additional repairs. Some of these additional repairs make intuitive sense and can be useful in expanding your thinking to some of the other viable possibilities. Many of the suggestions, unfortunately, don't make much sense. We are experimenting with heuristics to prune such unwanted suggested repairs. For instance, we protect some functions and predicates from repair, e.g., we do not allow either renaming or arity changing to, say, = or +.

8 Related Work

Several different fields of AI have investigated the problem of changing logical theories.

Belief revision: deals with the problem of adding a new belief to a theory that might result in it becoming inconsistent [Gärdenfors & Rott, 1995]. In this case, mechanisms have been designed to reestablish its consistency. These involve the identification of facts or (more rarely) rules to be deleted. Our work is complementary to this in suggesting changes to the *language* of the theory rather than the deletion of formulae.

Abduction: deals with the problem of identifying explanatory hypotheses for observations [Cox & Pietrzykowski, 1986]. New facts or rules are identified as hypotheses which, when added as new axioms to the ontology, enable the observations to be deduced. Again, our work is complementary to this in suggesting changes to the *language* of the theory rather than the addition of new formulae.

Ontology evolution: deals with the problem of managing an ontology[13]. Most of the work in this area consists of tools to assist manual construction and maintenance of ontologies. Our work differs in providing a tool for *automatic* identification of changes to the *language* of the ontology.

The IBIS system [Ekaterina Ovchinnikova, 2006] presents an approach for automatic ontology repair in the context of description logic. The approach focuses on repairing inconsistencies in terminological hierarchies. This approach detects three types of inconsistencies in these hierarchies: polysemy, single over-generalisation and multiple over-generalisation. Polysemy occurs when a concept is subsumed by two disjoint concepts. Single over-generalisation occurs when a concept is too specifically defined, so that one of its instances violates its definition. Multiple over-generalisation occurs when two or more definitions of the same concept conflict with each other.

The extended version of reformation for sorted logics [Mitrovic, 2013] suggests all the repairs made by the IBIS system, but also suggests additional sensible repairs. Reformation has, thereby, been shown to apply to description logic problems converted to first-order logic. This suggests that it will be possible to adapt reformation so that it applies directly to description logics.

Abstraction: has been frequently used in automated reasoning. For instance, [Giunchiglia & Walsh, 1989] surveys the uses of abstraction to simplify the original problem, solve the simplified problem, then map the solution of the simplified problem

[12] Actually, reformation also suggests removing the second argument from +, but we use heuristics to prune repairs to standard functions, such as +

[13] `http://www.w3.org/TR/webont-req/#goal-evolution`

to a solution to the original problem. In proposing theory repairs, our work uses both abstraction, e.g., merging functions, dropping arguments, and refinement, e.g., splitting functions, adding arguments. ORS explicitly based its repairs on the abstraction classification proposed in [Giunchiglia & Walsh, 1989], augmented with refinement techniques that inverted these abstraction techniques. Later work, e.g., reformation, has built on this classification and, thereby, provided a generative theory that underpins it.

Different logics: are used to formalise representations in different areas of research. Some common alternatives to first-order logic are:

Higher-order logics: which allow variables over functions, predicates, functions of functions, etc., as well as variables over domain objects. These variables can then be quantified. For instance, $\forall x, y.\ x = y \implies \forall p.\ p(x) \implies p(y)$ applies quantification to a unary predicate variable p. The work described in §5 represented both the theories and experiments of Physics and the ontology repair plans in higher-order logic.

Sorted and typed logics: which assign a type to each constant, variable and (sometimes) functions and predicates too. This is useful when the domain contains a mixture of different types of object, e.g., people, places, vehicles, etc. The work described in §5 used a typed, higher-order logic. Boris Mitrovic is currently extending reformation to various sorted first-order logics [Mitrovic, 2013].

Description logics: are restrictions of first-order logic so that reasoning always terminates. For instance, they usually only allow unary and binary predicates and no functions. The OWL family of description logics (OWL Lite, OWL DL, OWL2 and OWL Full) have been recommended by the World Wide Web Consortium as standards for representing knowledge represented on the internet [Bechhofer *et al*, 2004].

Although we have mainly described our work in terms of first-order logic, the discussion above shows that many of the techniques apply to these other logics too.

9 Conclusion

In this paper we have outlined how knowledge can be represented in logical theories and reasoning can be automated. We have emphasised the interaction of representation and reasoning: finding the right representation is the key to successful reasoning; failures of reasoning can suggest how faulty representations can be repaired. We have focused on two of the most common kinds of representational fault: a false conjecture can be proved; a true conjecture cannot be proved. We have illustrated the diagnosis and repair of faulty representations in three domains: multi-agent planning; physical theories and experiments; mathematical proofs. This work illustrated that it is possible not just to assert or retract beliefs represented as axioms, but that conceptual change is possible via language repair. Finally, we have discussed the reformation algorithm: a general-purpose algorithm for the diagnosis and repair of faulty representation languages.

Over the next half century, the automated repair of formal representations is set to be a key research area in automated reasoning and machine learning [Bundy & McNeill, 2006]. Until recently, most formal representations have been manually designed, by skilled knowledge engineers, for a narrowly defined reasoning application. If repairs were necessary, they have also been performed offline and manually. This situation is now changing:

- Automated agents are being used in applications where both the environment and the reasoning task are dynamically evolving. An agent's representation must also evolve to ensure its correctness and its suitability for the new problems it helps to solve.

- Multi-agent cooperation and competition are becoming more common and the number of agents involved is increasing, e.g., on the internet. Each of these agents may represent its knowledge of the environment differently, but the agents must communicate about the

environment they share. There are too many different representations for all the possible alignments to be anticipated, especially when they change faster than they can be manually aligned.

- The speed of agent interaction is increasing, e.g., in emergency response, so that manual repair or alignment is impractical. It must be automated, with provision for a human in the loop in some safety or security critical applications.

References

[Ascoli, 1883] Ascoli, G. (1883). Le curve limiti di una varietà data di curve. *Atti della R. Accad. Dei Lincei Memorie della Cl. Sci. Fis. Mat. Nat.*, 18(3):521–586.

[Baader & Snyder, 2001] Baader, F. and Snyder, W. (2001). Unification theory. In Robinson, J. A. and Voronkov, A., (eds.), *Handbook of Automated Reasoning, Volume 1*, volume I, chapter 8, pages 447–553. Elsevier.

[Bachmair & Ganzinger, 2001] Bachmair, L. and Ganzinger, H. (2001). Resolution theorem proving. In Robinson, J. A. and Voronkov, A., (eds.), *Handbook of Automated Reasoning, Volume 1*, volume I, chapter 2, pages 19–199. Elsevier.

[Bechhofer *et al*, 2004] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. and Stein, L. A. (2004). Owl Web Ontology Language 1.0 Reference. *W3C, Recommendation*.

[Bundy & McNeill, 2006] Bundy, A. and McNeill, F. (2006). Representation as a fluent: An AI challenge for the next half century. *IEEE Intelligent Systems*, 21(3):85–87.

[Bundy & Mitrovic,] Bundy, A. and Mitrovic, B. Reformation: A domain-independent algorithm for theory repair. In preparation.

[Bundy, 1985] Bundy, A. (1985). Poof analysis: A technique for concept formation. In Ross, P., (ed.), *Proceedings of AISB-85*, pages 78–86. Society for the Study of Artificial Intelligence and Simulation of Behaviour.

[Cox & Pietrzykowski, 1986] Cox, P. T. and Pietrzykowski, T. (1986). Causes for events: Their computation and applications. In Siekmann, J., (ed.), *Lecture Notes in Computer Science: Proceedings of the 8th International Conference on Automated Deduction*, pages 608–621. Springer-Verlag.

[Ekaterina Ovchinnikova, 2006] Ekaterina Ovchinnikova, and Kai-Uwe Khnberger and Tonio Wandmacher. (2006). Solving terminological inconsistency problems in ontology design. *IBIS – Interoperability in Business Information Systems*, 1.

[Fikes & Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

[Gärdenfors & Rott, 1995] Gärdenfors, P. and Rott, H. (1995). Belief revision. In Gabbay, D., Hogger, C. J. and Robinson, J.A., (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, pages 35–132. Oxford Science Publications, Oxford, New York.

[Giunchiglia & Walsh, 1989] Giunchiglia, F. and Walsh, T. (1989). Abstract theorem proving. In *Proceedings of IJCAI 89*, pages pp 372–377. International Joint Conference on Artificial Intelligence, Morgan Kaufman. Also available from Edinburgh as DAI Research Paper No 430.

[Gkaniatsou *et al*, 2012] Gkaniatsou, A., Bundy, A. and McNeill, F. (November 2012). Towards the automatic detection and correction of errors in automatically constructed ontologies. In *8th International Conference on Signal Image Technology and Internet Based Systems*, pages 860–867, Sorrento, Italy. IEEE.

[Lakatos, 1976] Lakatos, I. (1976). *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press.

[Lehmann *et al*, 2012] Lehmann, J., Chan, M. and Bundy, A. (2012). A higher-order approach to ontology evolution in physics. *Journal on Data Semantics*, pages 1–25.

[McCarthy, 1964] McCarthy, J. (1964). A tough nut for proof procedures. Stanford Artificial Intelligence Project Memo 16, Stanford University.

[McNeill & Bundy, 2007] McNeill, F. and Bundy, A. (2007). Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. *International Journal On Semantic Web and Information Systems*, 3(3):1–35. Special issue on ontology matching.

[Mitrovic, 2013] Mitrovic, B., (2013). Repairing inconsistent ontologies using adapted reformation algorithm for sorted logics. UG4 Final Year Project, University of Edinburgh.

[Pólya, 1945] Pólya, G. (1945). *How to Solve It*. Princeton University Press.